PLASTIC

IST STREP Project

Deliverable D3.3.4
Middleware User & Developer Guide on *Service Discovery*

http://www.ist-plastic.org

Project Number	:	IST-26955
Project Title	:	PLASTIC
Deliverable Type	:	Report

Deliverable Number	:	D3.3.4
Title of Deliverable	:	
Nature of Deliverable	:	User and Developer Guide
Dissemination level	:	
Internal Document Number	:	
Contractual Delivery Date	:	
Actual Delivery Date	:	
Contributing WPs	:	
Editor(s)	:	
Author(s)	:	Pierre-Guillaume Raverdy
Reviewer(s)	:	

Abstract

Service discovery is an essential function of SOC as it enables the runtime association to the networked services. Three basic roles are identified for service discovery: (i) Service provider is the role assumed by a software entity providing a networked service; (ii) Service requester is the role of an entity seeking to consume a specific service; (iii) Service repository is the role of an entity maintaining information on available services and a way to access them. Building on the tremendous number of proposed service discovery protocols and accounting for the specifics of pervasive computing, we introduce the Plastic Service Discovery Service that provides dynamic, interoperable context-aware service discovery. This discovery service leverages B3GSOAP to support service discovery in multi-radio, multi-network environments

Keyword list

Service discovery, interoperability, multi-network discovery

Document History

Version	Type of change	Author(s)
1.0	Initial published version	PG Raverdy

Document Review

Date	Version	Reviewer	Comment
<date of review>	<version>	<name & affiliation>	<proofed, found weaknesses, corrections …>

Table of Contents

1 Component Overview 4

1.1 PLASTIC Service Discovery Service 5

1.1.1 Service description and matching 6

1.1.2 Plastic SD Service Components 7

1.2 Software Information 8

1.2.1 Language 8

1.2.2 Provider 9

1.2.3 System requirements 9

1.2.4 Download 9

1.2.5 Set-up information 9

1.2.6 License 10

1.3 Software package 10

1.3.1 Content 10

1.3.2 Related documents 11

1.3.3 Development status and limitations 11

1.3.4 Future developments 11

2 Software deployment and use 13

2.1 Deployment procedure 13

2.1.1 Deploying the PLASTIC SDS in Tomcat 13

2.1.2 Deploying the PLASTIC SDS as a Stand-alone application 13

3 Component Architecture 14

3.1 Java Packages Overview 14

3.2 The Plugin Manager component 15

3.3 The Storage Manager component 16

3.4 The Discovery Manager component 17

3.5 The PLASTIC Discovery Service component 18

3.6 The PLASTIC Group Discovery Service component 18

1 Component Overview

B3G networks combine multiple wireless networking technologies in order to benefit from their respective advantages and specificities. Further, the increase in computing and communication capacities of portable devices, as well as their mass marketing, makes possible the widespread deployment of such multi-networks pervasive environments, where B3G-capable devices hold several radio interfaces (e.g., UMTS, WiFi, Bluetooth), and the possibility to switch from one radio interface to another. PLASTIC users shall then benefit from such an environment by increasing the perimeter of reachable service providers. However, this should not be realized at the expense of a greater complexity. Figure 1 illustrates a B3G environment with users in a shopping mall being able to connect to various wireless networks deployed there (a MANET, a hotspot and a Bluetooth PAN) and being able to connect to their intranets or home networks through Internet or cellular networks, possibly exploiting bridging capabilities of the networked nodes (e.g., MANET node accessing the Internet via the cell phone).

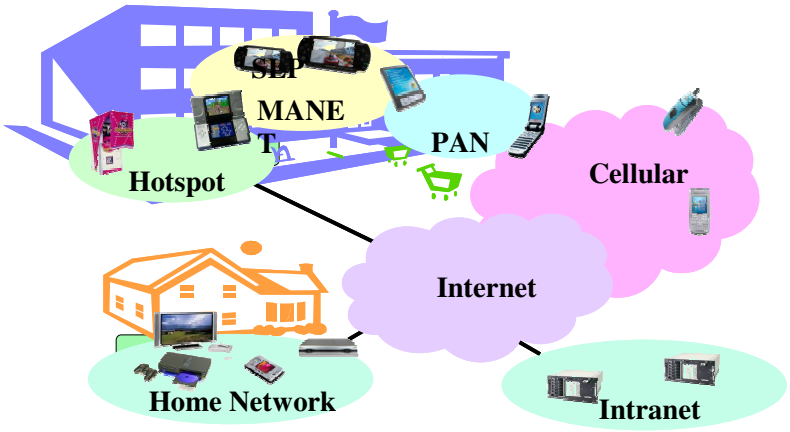

Figure 1: B3G environment

In order to offer B3G users a variety of application services exploiting the network's diversity and richness, programmers should be provided the relevant abstractions for managing both the use of multi-radio networks and their context, and these abstractions should be consistently integrated with the software architectural model on which the application is based.

Service Oriented Architectures (SOA) is a loosely-coupled model whose most significant aspect is that it separates the service implementation and the service provider from its contractual description. Web Services (WS) represent a concrete instantiation of SOA. Web Services are components that can be integrated into more complex distributed applications by means of some Web-based/XML-based open standards (e.g. WSDL, UDDI, HTTP, SOAP, etc.).

Service discovery is an essential function of SOA as it enables the runtime association to the networked services. Three basic roles are identified for service discovery: (i) Service provider is the role assumed by a software entity offering a networked service; (ii) Service requester is the role of an entity seeking to consume a specific service; (iii) Service repository is the role of an entity maintaining information on available services and a way to access them. A service description formalism or language to describe the functional and non-functional properties (such as QoS, security or transactional aspects of networked services) complemented with a service discovery protocol enables service providers, requesters and repositories to interact with each other. Many academic and industry supported SDPs have already been proposed

and leading SDPs in dynamic environments use a pull-based approach (SLP, WS-Discovery, Jini, SSDP), often supporting both the centralized and distributed modes of interaction: clients send requests to service providers (distributed pull-based mode) or to a third-party repository (centralized pull-based mode) in order to get a list of services compatible with the request attributes.

Building on the tremendous number of proposed service discovery protocols and accounting for the specifics of pervasive computing, we introduce the PLASTIC Service Discovery (PSD) Service that provides dynamic, interoperable, context-aware service discovery. The PSD service is mainly a reengineering of the open source MUSDAC multi-protocol service discovery platforms on top of B3GSOAP in order to support service discovery in multi-radio, multi-network environments. While in MUSDAC service discovery and access were tightly integrated, the PSD service is only concerned with service discovery, and relies on B3GSOAP group communication to disseminate service information across networks. Changes in the multi-network topology (e.g., broken propagation paths) are then taken care of transparently.

1.1 PLASTIC Service Discovery Service

The overall architecture of the PLASTIC middleware solution for service discovery in the B3G networking environment is depicted in Figure 2. Compared to traditional service discovery solutions, the PLASTIC service discovery functionality promotes interactions with networked services that match required functional properties while accounting for the context and QoS of service provisioning (i.e., from both the provider and consumer perspectives). This leads us to introduce a modular service description (See Section 1.1.1), which abstractly characterizes the context and QoS of networked services. Service discovery within a network is coordinated by a (logically) centralized service (the *PLASTIC* SD Service), also known as *repository*. The behavior of PSD service is further detailed in Section 1.1.2.

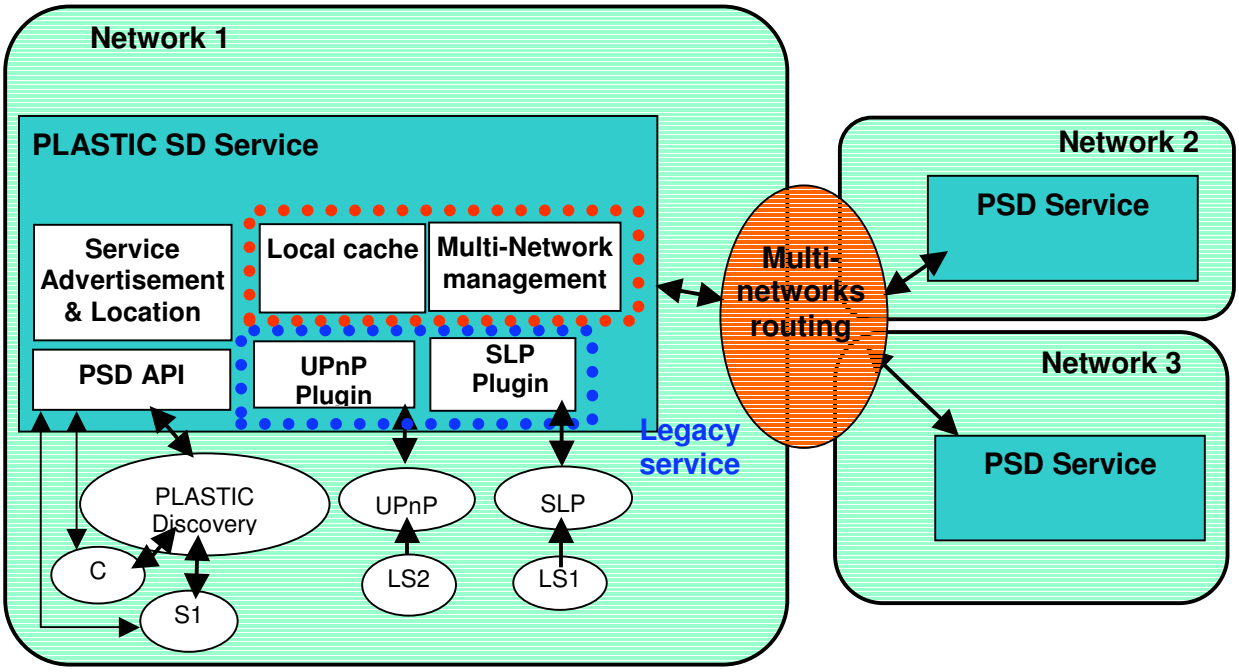

Figure 2: B3G service discovery within the PLASTIC middleware

1.1.1 Service description and matching

In most service discovery protocols, service descriptions used for service discovery generally describe, in a single description, the service's functional behavior in terms of offered operations. In PLASTIC, we strive to take into account both functional (e.g., methods) and non-functional (e.g., QoS, context) properties, and cater for future/alternative description formats.

We therefore introduce an extensible, composite service description model for PLASTIC services, which is to be used by both client (for service location) and server (for service advertisement) in the service discovery process (See Figure 3). More precisely, we define a generic container that contains a single description. A complete service description is composed of one or more containers, each containing a different descriptions format (i.e., each representing a facet of the service) such as supported methods, QoS properties, context, mobility model…. All containers associated to a service are directly linked to a root/top-level container (1-level hierarchy). Each container contains the unique ID of the root element (for composition/linking purposes), the unique ID of the element, the name and type of the element, and finally a string containing the actual description in the relevant format. The actual description content is embedded in the container and needs to be parsed by the PSD service in order to process it.

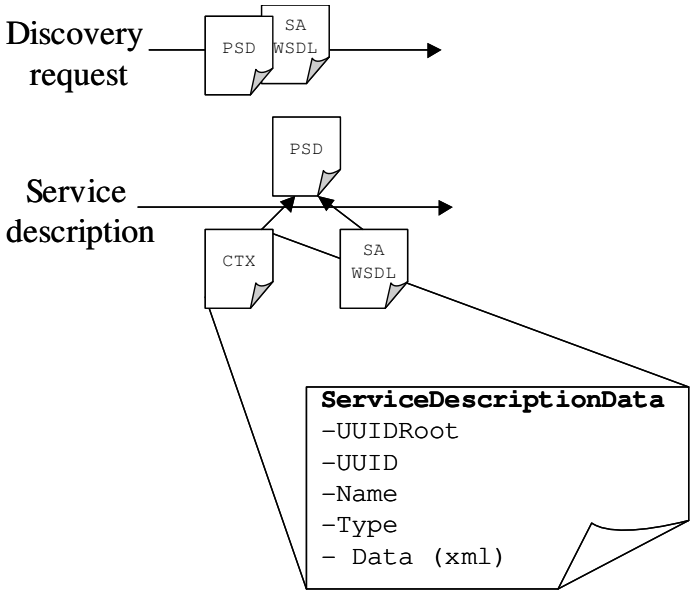

Figure 3: PLASTIC Composite Service Description

Mobility model

Various approaches to the management of (physical) mobility in the context of WS access can be envisioned. Those approaches specifically relate to handling connectivity loss with the host of a given networked service instance with which a session is ongoing. Solutions basically subdivide into either relocating the service's host or substituting the service instance with another offering a *matching* behavior. The advantage of the former is that seamless mobility is achieved, avoiding failures due to connectivity loss; however, this may be at the expense of performance regarding in particular response time. The advantage of the latter approach is that service substitution may be realized so that mobility does not impact performance; however, this requires management of system state consistency, regarding both the service's client and provider.

In general, mobility shall be managed according to the application specifics, as recognized since early work on nomadic computing. Thus, the abstract description of services, as used in the service discovery process to both advertise and locate networked service instances, must be enriched with mobility awareness attributes, guiding mobility management at the middleware layer. Such attributes relate in particular to the mobility patterns of nodes together with the quality of network connectivity so as to be able to probabilistically estimate the duration of connectivity between client and provider.

Context & QoS specification

Context and QoS awareness is key to service provisioning in the B3G networking environment. Service descriptions shall then provide relevant information to be used at the time of service discovery:

- Context-related requirements may be specified within the service description, building upon the PLASTIC middleware support for context management.
- QoS-related specification within service description shall be derived from related Service Level Specification used at the time of application software design and analysis, based on the SLAng language.

1.1.2 Plastic SD Service Components

As depicted in Figure 2, the PLASTIC SD service decomposes into a number of functionalities:

- The service advertisement & location component allows for publishing and locating services in the B3G network.
- The PLASTIC SD API provides an enhanced interface for service discovery with respect to the handling of context and QoS of service provisioning.
- Locating non-PLASTIC services using legacy SDPs is supported through plugins.
- Thanks to the underlying PLASTIC multi-network communication, service discovery can span over loosely interconnected heterogeneous networks.
- Dedicated support is integrated towards mobility management so as to offer mobility-aware service discovery.

The above functionalities are further detailed below.

Service advertisement and location

The base service advertisement and location protocol of the PLASTIC SD service provides an explicit discovery an registration API to account for both functional and extra functional properties of services (mobility, context), and builds upon the MUSDAC middleware to enable interoperability with legacy service discovery protocols.

The PLASTIC service discovery functionality is semi-distributed, being structured around the PSD-service that act as a service registry within a network. By default, we assume that there is one PSD-service per network, although it may be physically distributed for scalability issues.

Advanced service discovery features

The PLASTIC service discovery process shall take into account the specific features of the B3G networking environment, in particular calling for awareness of mobility, context and QoS of the service provisioning. Accordingly, service advertisement and request formats in PLASTIC embed enriched service descriptions, as introduced in Section 1.1.1. A requested service is then located and selected according to the matching relationships and selection policy specified by the application..

In order to support seamless mobility, a service request may specify the specific host of a service using the host's PLASTIC@. In this case, the service location amounts to locate the given node, as opposed to retrieving any networked service instance matching an abstract service description.

Multi-protocol service discovery

Multi-protocol service discovery is achieved using service discovery plugins (e.g., UPnP and SLP plugins) so as to collect and convert SD-specific service descriptions. Depending on the SDP, the plugin either registers for service advertisements (i.e., push-based protocol), or directly performs service discovery (i.e., pull-based protocol). In the case of push-based protocols, the PLASTIC service description generated from the service announcement is recorded it in its local repository (i.e., *local cache*). For these push-based protocols, the PSD-service directly evaluates PLASTIC discovery requests against the repository (i.e., without forwarding it to the plugins), and returns the previously generated service descriptions.

Multi-network service discovery

The PLASTIC middleware allows for communication over a path of heterogeneous networks thanks to PLASTIC multi-network routing. In particular, PLASTIC service discovery may traverse the networks that are loosely coupled by the networked bridges. Specifically, we introduce the *B3GSOAP Discovery group* over the PLASTIC multi-network multi-radio multicast routing, which enables the dissemination of service requests across networks (i.e., to the PSD-Services embedded in those networks). We consider only pull-based multi-network service discovery as a first tradeoffs between the communication cost and the flexibility of pervasive networking.

Mobility management

In general, mobility management is best addressed though various complementary approaches at the middleware layer. PLASTIC service discovery (together with the tracking of networks IDs and IP addresses associated to a PLASTIC@ achieved in the Multi-Radio Multi-Networks communication layer) is a central element in the management of mobility within the PLASTIC middleware. Indeed, it serves both relocating a given service and locating a valid substitute for a service. Furthermore, the service registry may track the mobility of nodes in the background. Indeed, the PSD-Service may try maintaining accurate network addresses for service descriptions stored within the local cache and known to be accessed by some local client(s). Such a feature is basically driven by the information embedded within service descriptions and does not raise any major difficulty. However, a challenging issue for the service discovery process is to locate and select service instances according to the respective mobility patterns of client and provider, so that it favors interaction between nodes that will remain in network reach of each other for the duration of the session. Further, in the PLASTIC multi-radio, multi-networks environment, network reachability may span various radio networks and/or multiple network hops during a single session, and possibly involve the relocation of the service's host. Such an issue is part of our ongoing work, where adequate tradeoffs need to be established between the cost of managing mobility patterns and the base performance of mobility management simply relying on the underlying network.

1.2 Software Information

1.2.1 Language

Java J2SE 1.4.2 or above running on any Operating System

1.2.2 Provider

INRIA-Rocquencourt

1.2.3 System requirements

In order to use PLASTIC SDS, the following components are required:

1. J2SE 1.4.2 JDK or above
2. Latest Eclipse with Web development tools (for generating WS stubs and webapps)
3. Apache Axis2 (v1.2) libraries and the Eclipse plugin

   http://ws.apache.org/axis2/1_2/contents.html
4. Apache commons packages (i.e., lang, configuration, collection)

   http://commons.apache.org/collections/

   http://commons.apache.org/configuration/

   http://commons.apache.org/lang/
5. (optional) Apache Tomcat v5.5
6. (optional) PLASTIC's Multi-radio networking library and B3GSOAP

The installation procedures for these software packages can be found in their documentation.

1.2.4 Download

The PLASTIC Service discovery service, B3GSOAP and Multi-radio networking packages can be downloaded from the IST PLASTIC Website (http://www.ist-plastic.org).

In the section Workpackage / WP3 / WP3 Software

1.2.5 Set-up information

In order to access the PLASTIC SDS (clients or services), the following components are required.

- Java J2SE virtual machine (v1.4.2 or higher)
- Axis2 libraries and WS-addressing handler extension
- Apache commons libraries (i.e., lang, configuration, collection)
- UDDI4J, Open SLP and Cyberlink UPnP libraries
- B3GSOAP library
- (optional) PLASTIC's Multi-radio networking library (see user's manual for installation)
- (optional) Device with multiple network interfaces

Important Notes:

- Axis2 v1.2 (April 2007) is currently supported. Newer versions are not backward compatible.
- Tomcat version 5.5 has been used for experiments. The configuration files for Tomcat are quite different between versions 4, 5, and 6. The deployment information provided in this section is provided for Tomcat 5.5. For other versions, the configuration may or may not be different.
- It is recommended to install Eclipse (and the Axis2 plugin) in the Windows harddisk/partition. If not, some errors (such as unknown class exception) may be reported when generating the stubs for the Web service.

1.2.6 License

Copyright (C) 2006-2008 PLASTIC CONSORTIUM[1]

The PLASTIC SDS library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details[2].

In addition, the following external libraries (aunder their respective licence) are used in PLASTIC SDS:

Component	Software used	Context	Licence	Replacement
B3GSOAP	Apache Axis2 http://ws.apache.org/axis2/	embedded in Bridge and in stand-alone services	Apache license v2	Need to port code for B3G transports and fir the bridge on a different SOAP engine
	Apache Tomcat http://tomcat.apache.org/	For deploying B3G Web services	Apache license v2	Use another application server (i.e., port B3G transports code on another SOAP engine)
PLASTIC SD	UDDI4J http://uddi4j.sourceforge.net/	Used in the UDDI plugin for SD	IBM Public License.	Find another client library for UDDI and port the plugin
	OpenSLP http://www.openslp.org/	Used in the SLP plugin for SD and for SLP test services	BSD License	Find another client library for SLP and port the plugin/test services
	Cyberlink for Java http://sourceforge.net/projects/cgupnpjava/	Used in the UPnP plugin for SD and for UPnP test services	BSD License	Find another client library for SSDP/UPnP and port the plugin/test services

1.3 Software package

1.3.1 Content

The PSDS software package described in this documentation provides:

1. Implementations of an enhanced SD protocol, which can be used by clients to discover services in the B3G environment.
2. A set of legacy SD plugins to discover legacy services in the B3G environment.
3. Simple test applications to demonstrate the registration and discovery of a service

[1] http://www.ist-plastic.org

[2] http://www.gnu.org/licenses/lgpl.html

The PSDS Software package includes the following files:

```
PSDSDIR
      |
      + bin            (plastic sds jar)
      + manuals        (this guide)
      + doc            (Javadoc API)
      + src            (source code)
      + WebContent     (configuration for Tomcat deployment)
      + License        (LGPL Licence Agreement)
```

1.3.2 Related documents

The following documents are related to PLASTIC Service Discovery Service

- PLASTIC Deliverable 3.1
- PLASTIC Deliverable 3.2 (source code and report)
- B3GSOAP User Manual

1.3.3 Development status and limitations

The current version of the B3GSOAP software package is version 1.0. The status of the provided component is as follows:

- The SLP, UPnP and UDDI legacy SDP are supported, Service description formats supported are WSDL/SAWSDL
- The DiscoveryService supports both synchronous and asynchronous discovery requests. For both methods, a timeout must be provided. For synchronous calls, the matching results are returned after the timeout while in the asynchronous case, the results are dropped after the timeout (hence, the client application should fetch the results before the end of the timeout.
- Mobility management is not yet implemented.
- B3GSOAP limitations apply for multi-network service discovery.
- The PSD-Service can currently be deployed only on top of the Axis2 SOAP engine, and client/service application stubs for the PLASTIC discovery service and only provided for Axis2.

1.3.4 Future developments

The following developments may be undertaken in order to improve B3GSOAP

- **QoS aware multi-network routing protocol**: the current multi-network routing protocol implemented in the Bridge component selects the network route based on the shortest path. Alternative routing protocols taking into account the QoS of the different networks and/or the capabilities of the Bridges (cpu, memory, load) should be provided.
- **Mobility support:** the current version of the B3GSOAP transports and Bridge do not handle changes in the network interfaces (up/down, change of IP address, …). These components should therefore be updated to handle both client and service mobility. Such handling also requires to provide a scheme for re-discovering the mapping

between PLASTIC@ and the set of IP addresses that may involve the multi-radio networking layer, B3GSOAP and PLASTIC's service discovery service.

2 Software deployment and use

Once downloaded the file plastic-sds-v1_0.zip:

1. Unzip the file into a directory (e.g., C:\plasticsds)
2. In the directory you unzipped the file, there should be the following subdirectories:

```
bin, src, doc, manuals, licence
```

2.1 Deployment procedure

In this section, we describe how to deploy the PLASTIC SDS, register a service, and how to start a client application accessing the service.

2.1.1 Deploying the PLASTIC SDS in Tomcat

In order to deploy the PLASTIC SDS as a Web Service in Tomcat, the following steps should be performed:

- The B3GSOAP library should be installed and configured (See B3GSOAP documentation).
- In the PLASTIC SDS source code, the servlet code in should be un-commented and the project recompiled
- The PLASTIC SDS code should be exported as a war file, the servlet must be installed in the `webapps` folder of Tomcat, and the configuration files (web.xml) properly configured

2.1.2 Deploying the PLASTIC SDS as a Stand-alone application

In order to deploy the PLASTIC SDS as a stand-alone Java application, the following steps should be performed:

- The B3GSOAP library as well as the Axis2 and other apache libraries should be in the classpath
- The B3GSOAP configuration file should setup and accessible (see B3GSOAP documentation)
- The PLASTIC SDS project should be exported as a jar file
- The `DiscoveryServiceApp` main class in the `org.plastic.sd.sds` package should be started

3 Component Architecture

Figure 3 presents a refined architecture of PLASTIC SDS. Its components are:

- The **Plugin Manager** that controls **SDP Plugins** which interact with clients and services using legacy SD protocols (e.g., UDDI, SLP, SSDP);
- The **PLASTIC Discovery Service** that enables PLASTIC-aware applications to directly register or discover services in the B3G environment using composed service descriptions.
- The **Group Discovery Service** that route discovery requests and responses between PSD services in the B3G network using B3GSOAP;
- The **Storage Manager** that maintains specific **Stores** for the various facets of the service description (e.g., Plastic address, context, SSDP service description, SAWSDL service description); Each store is able to parse its associated service description format and to perform matching operations between service descriptions of this format.
- Finally, the **Discovery Manager** that starts the other components and controls on-going requests processed by local plugins or by remote PLASTIC SDS.

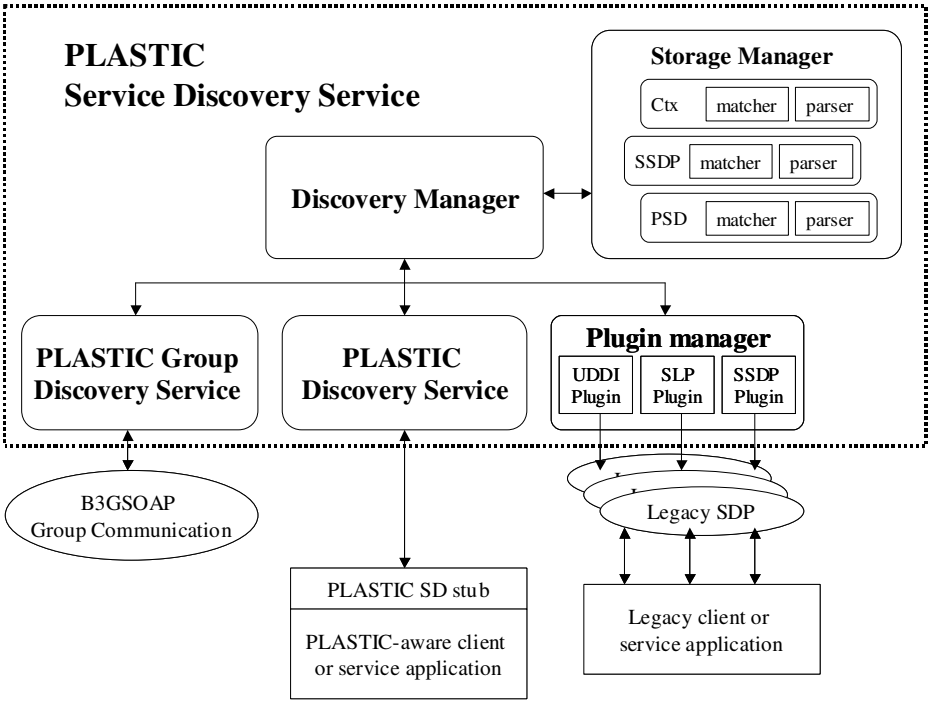

Figure 3: PLASTIC SDS Components

We now detail the different Java packages available in the source folder, and then detail each of the above

3.1 Java Packages Overview

The Java source folder in the B3GSOAP project contains the following Java packages:

- `org.plastic.sd.descr.*`: this package contains several sub-packages that deal with the representation of the various facets of a service description. For each facet's description format (PSD, SLP, SSDP, PADDR, SAWSDL) a class implements the

`DescriptionInterface` interface that will be used by the storage and matcher components,

- `org.plastic.sd.manager`: this package provides classes for the core `DiscoveryManager` component,
- `org.plastic.sd.plugins.*`: this package contains several sub-packages that provide specific plugins for legacy SD protocols.
- `org.plastic.sd.sds`: this package contains the Axis2-specific implementations of the `DiscoveryService` (for client and service applications) and the `GroupDiscoveryService` (for multi-network discovery between PSD services).
- `org.plastic.sd.storage.*`: this package contains several sub-packages that deal with the parsing and matching of specific facets of a service description. It also contains the `StorageManager` class that records individual storage and matching classes for the supported formats, and interacts with the discovery manager to filter the matching descriptions for a request.
- `org.plastic.sd.test`: this package contains a test application for service registration and client discovery.

3.2 The Plugin Manager component

The *PluginManager* component abstracts the legacy SD protocols. It registers and activates legacy SD plugins provided by the *DiscoveryManager*, and supports synchronous service discovery requests (asynchronous service notification from SD plugins is not yet supported), The *`PluginManager`* component interface supports

- Activation of a plugin
  - `public void addPlugin ( int sdpType, PluginInterface plugin )`
- Desactivation of a plugin
  - `public void delPlugin ( int sdpType )`
- Discovery of services
  - `public Vector<ServiceDescriptionData> getServices ( ServiceDescriptionData[] serviceDescriptions )`
  - where the `ServiceDescriptionData` elements describing facets of the requested service are provided.

For each discovery request forwarded by the *DiscoveryManager*, the *PluginManager* sequentially calls each active SD Plugin with the set of `ServiceDescriptionData` provided. Currently, each call is performed sequentially but could be processed in parallel by different threads, as each protocol is independent (no side effects).

Each legacy SD plugin implements the `PluginInterface` interface that supports

- Discovery of services
  - `public Vector<ServiceDescriptionData> getServices ( ServiceDescriptionData[] serviceDescriptions )`

Each SD plugin is in charge of translating the request's `ServiceDescriptionData` into the appropriate legacy SD request format, perform the request, translate the matching service descriptions into the appropriate sets of `ServiceDescriptionData` and return them to the

PluginManager. The *PluginManager* then merges all the responses and returns them to the *DiscoveryManager*,

3.3 The Storage Manager component

The *StorageManager* is in charge of storing service descriptions from PLASTIC service registrations, SD Plugins announcements, or remote discovery responses. It is also in charge of returning the stored service descriptions matching a given request. As depicted in Figure 3, a PLASTIC service description (or request) is composed of a set of elementary descriptions representing functional and extra-functional properties. Therefore, the *StorageManager* relies on a set of *SingleStorage* components able to parse and match specific elements of a service description (See Figure 4). The `SingleStorage` class is generic and new description element formats are readily supported by providing the appropriate `Matcher` and `Parser` components, In order to discover the matching services to a request, the StorageManager component calls the SingleStorage components associated to each element of the discovery request, and gets the IDs of the services that match this element. The list of matching services is passed on to the next SingleStorage in order to refine the list of services that match all elements in the request.

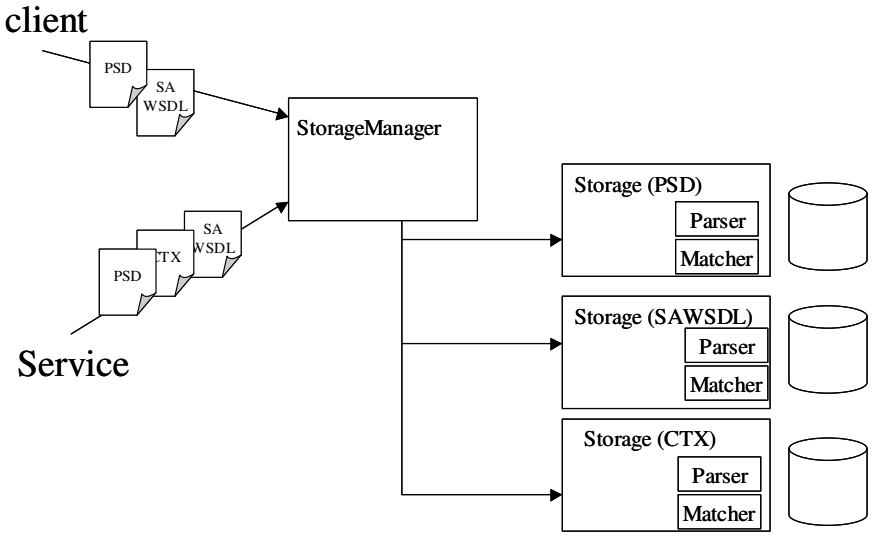

Figure 4: Storage Manager Component

The StorageManager implements the `StorageInterface` *that* supports

- Adding a `ServiceDescriptionData` element (part of a service description)
  - **`public boolean`** `add (ServiceDescriptionData serviceData)`
- Removing the `ServiceDescriptionData` element associated to a given UUID, removing also all the sub-elements if this element is the root of the service description
  - **`public boolean`** `remove (String uuid,` **`boolean`** `isRoot)`
- Retrieves all the `ServiceDescriptionData` elements associated to the given UUIDs
  - **`public`** `Vector<ServiceDescriptionData> get (Vector<String> uuids)`
- Returns all the UUIDs of the services that match the provided `ServiceDescriptionData` element. This element can be of any type.
  - **`public`** `Vector<MatchResultGeneric> findMatch (ServiceDescriptionData serviceData)`

- Returns the UUIDs of the services that match the provided `ServiceDescriptionData` element and that belong to the provided UUID set. This call allows to filter the UUIDs of services that match a set of `ServiceDescriptionData` elements element can be of any type,
  - **public** `Vector<MatchResultGeneric> findMatch (ServiceDescriptionData serviceData, Vector<String> subsetRootUUIDs)`

Each *SingleStorage* implements the *`StorageInterface`* and manages `ServiceDescriptionData` elements of a single type. Each *SingleStorage* contains a Parser (for unmarshalling and marshalling the data in the `ServiceDescriptionData` element into/from Java objects) and a Matcher (for comparing two objects of the supported type)

The `ParserInterface` supports the following methods (self-describing)

- **public boolean** `isvalidDescription ( String xmlData );`
- **public** `Object parseDescription ( String xmlData );`
- **public** `String ouputDescription ( Object description );`

The `MatcherInterface` supports the following method (self-describing)

- **public boolean** `match (Object descr1, Object descr2);`

3.4 The Discovery Manager component

The *DiscoveryManager* component is the central entity of the PLASTIC SD service. It is the first component to be started, and is in charge of starting all the other components (plugins, storage based on the service configuration).

The *DiscoveryManager* implements the `DiscoveryManagerInterface` that provides the following operations

- Registration of applications (in fact both client and service applications) and generation of an authorization code for registering, updating, and removing services or for issuing requests.
  - **public** `String registerClient ( java.lang.String userid, java.lang.String cred);`
- The registration, update, and removal of service descriptions
  - **public boolean** `registerService ( java.lang.String authInfo, ServiceDescriptionData[] serviceDescriptions, Integer lifetime );`
  - **public boolean** `updateService ( java.lang.String authInfo, ServiceDescriptionData[] serviceDescriptions, Integer lifetime );`
  - **public boolean** `unregisterService ( java.lang.String authInfo, String serviceUuid );`
- The synchronous discovery of services matching a specific set of `ServiceDescriptionData` elements.
  - **public** `ServiceDescriptionData[] getServices ( java.lang.String authInfo, ServiceDescriptionData[] serviceDescriptions );`
- The asynchronous discovery of services matching a specific set of `ServiceDescriptionData` elements and the retrieval of the matching services,
  - **public int** `searchServices (java.lang.String authInfo, ServiceDescriptionData[] serviceDescriptions,` **int** `timeout, String dissemination );`
  - **public int** `getSearchStatus (java.lang.String authInfo, Integer requestId);`
  - **public** `ServiceDescriptionData[] getSearchResponse (java.lang.String authInfo, Integer requestId, Boolean getAll, Boolean cleanAll);`
- The retrieval of a complete service description based on its UUID
  - **public** `ServiceDescriptionData[] getService(String authInfo, ServiceDescriptionData serviceDescription);`
- The retrieval of the updated `ServiceDescriptionData` elements of a service

- `public ServiceDescriptionData[] getUpdate (String[] descriptionUuids, Integer versionId);`

The *DiscoveryManager* also implements the `RemoteDiscoveryInterface` for multi-network discovery support. To achieve this, the DiscoveryManager of a PSD-service sends a discovery request using the B3GSOAP discovery group. The request is then received by PSD-services in the B3G environment that processes it locally. Each PSD-service returns the matching descriptions back to the initiator also using the B3GSOAP discovery group. All responses are collected by the initial DiscoveryManager and returned to the client application. The `RemoteDiscoveryInterface` interface provides the following callback operations:

- Remote discovery request received on the B3GSOAP discovery group to be processed locally.
  - `public void newRemoteRequest (int requestId, String uuidSender, ServiceDescriptionData[] request, int timeoutSec);`
- Remote discovery response received only by the initiator of the request with the remote service descriptions to be returned to the client application.
  - `public void newRemoteResponse (int requestId, ServiceDescriptionData[] response);`

3.5 The PLASTIC Discovery Service component

The DiscoveryService is a simple class deployed as a Web Service on top of the SOAP engine use (i.e.. Axis2 in the current implementation), which implements the `DiscoveryManagerInterface` and acts as a wrapper for the DiscoveryManager singleton (i.e., for each client/service application's session, a DiscoveryService WS instance is created. This instance then redirects incoming requests to the DiscoveryManager). The WS stubs for the DiscoveryService are used by both client and service applications to register services or perform discovery requests.

3.6 The PLASTIC Group Discovery Service component

Similarly, the GroupDiscoveryService is a simple class deployed as a Web Service on top of the SOAP engine (i.e., Axis2). It implements the `RemoteDiscoveryInterface` and acts as a wrapper for the DiscoveryManager singleton. When a PSD-Service instance is started, the Axis2 SOAP engine is configured with both B3GSOAP point-to-point and group transports. The GroupDiscoveryService service is instantiated and configured with the Discovery group PLASTIC@ (i.e., it uses the B3GSOAP group transport).

The WS stubs for the GroupDiscoveryService are used by DiscoveryManager of PSD-service instances to perform discovery requests in the B3G environment. This service is not to be accessed by client or service applications.